**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №3

дисциплина: ЭВМ и периферийные устройства

тема: «Изучение принципов организации обмена данными по

последовательному интерфейсу I2C на примере управления

блоком светодиодов и программного опроса клавиатуры»

Выполнил: ст. группы ВТ-32

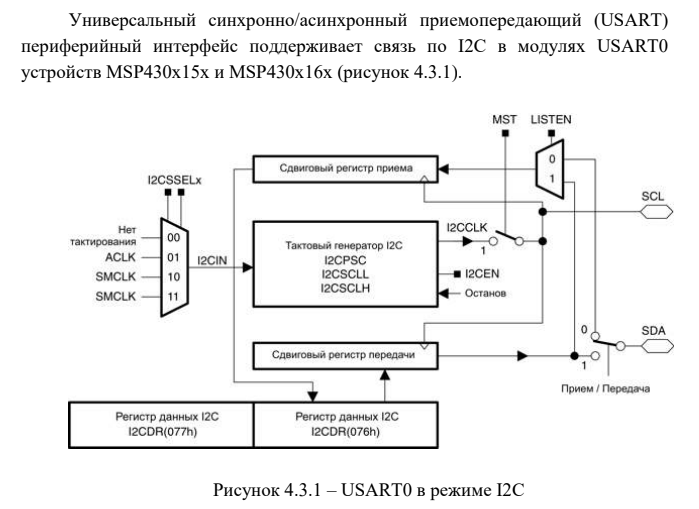
Воскобойников И. С.

Проверил: Шамраев А.А.

Белгород 2021

**Цель работы**: Изучить принципы программного управления двунаправленным обменом данных по последовательному интерфейсу I2C.

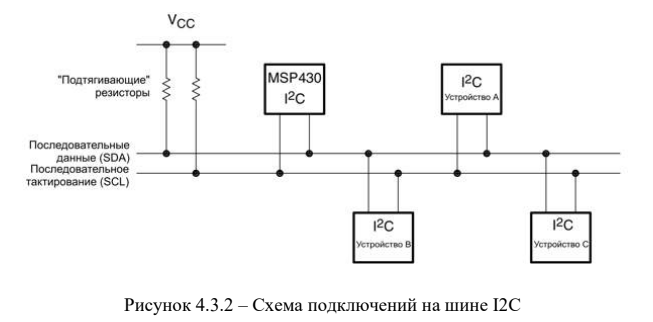
**Теоретические сведения**

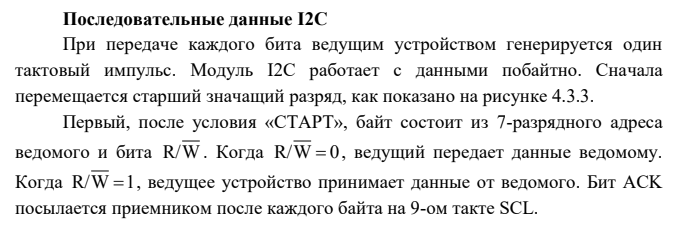
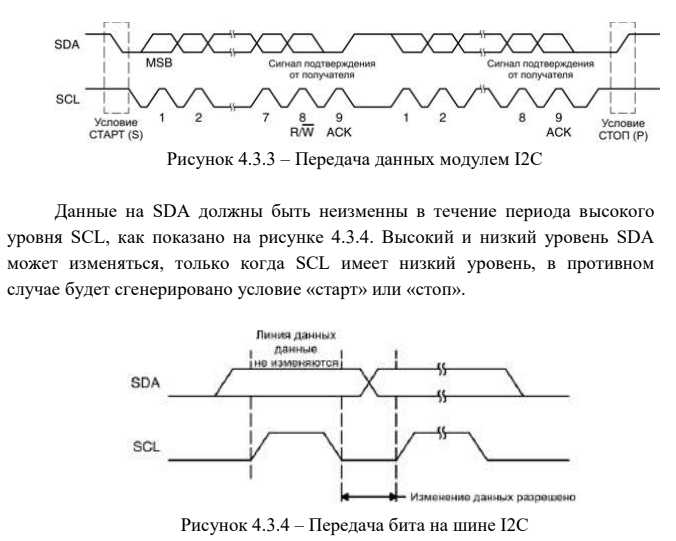


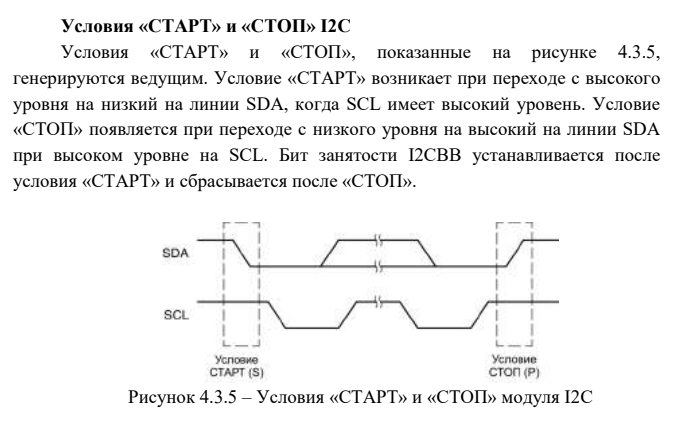
Модуль управления взаимообменом между интегральными схемами (I2C) обеспечивает интерфейс между MSP430 и I2C-совместимыми устройствами через последовательную двухпроводную шину I2C. Внешние компоненты, подключенные к шине I2C, последовательно передают и принимают последовательные данные в/из USART через 2-х проводной I2C-интерфейс.

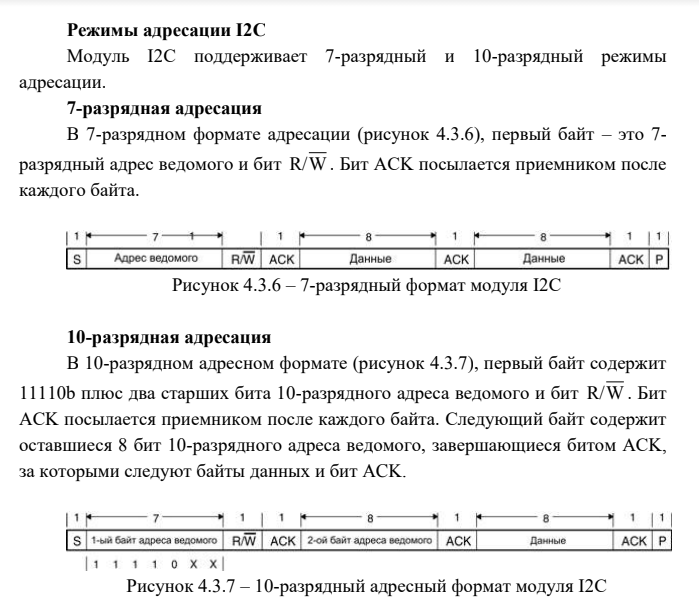
**Функционирование модуля I2C**

Модуль I2C поддерживает любые ведущие и ведомые устройства, совместимые с I2C. На рисунке 4.3.2 показан пример шины I2C. Каждое устройство обладает уникальным адресом и может работать и как передатчик и как приемник. Устройство, подключенное к шине I2C, во время передачи данных может рассматриваться как ведущее или ведомое. Ведущий инициирует передачу данных и генерирует тактовый сигнал SCL. Любое устройство, адресованное ведущим, рассматривается как ведомое.

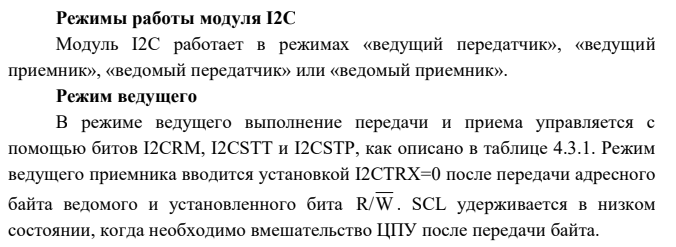


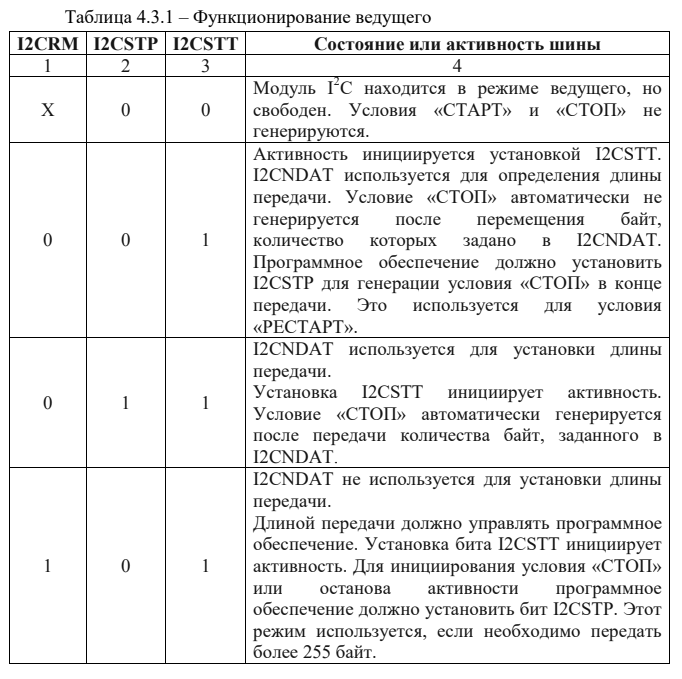
**** 

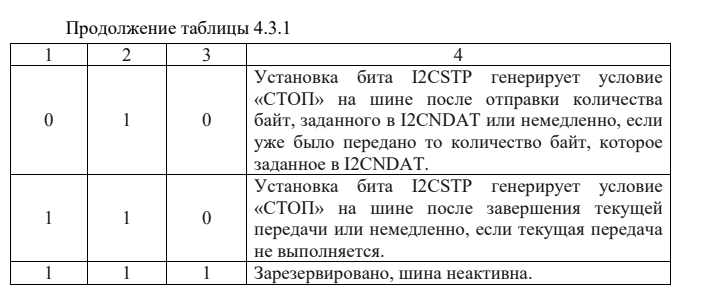


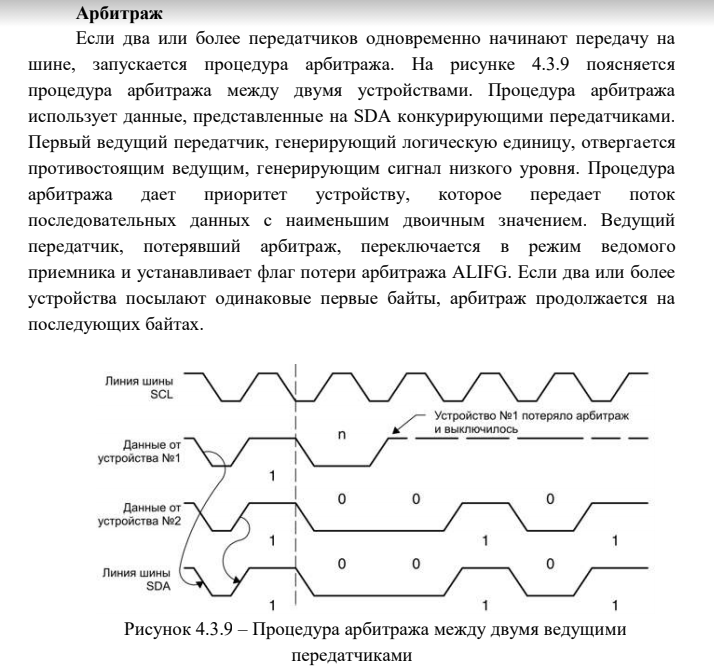












Если выполняется процедура арбитража, когда на SDA повторяются условия «СТАРТ» или «СТОП», ведущие передатчики, вовлеченные в арбитраж, должны послать повторные условия «СТАРТ» или «СТОП» в том же самом месте в формате фрэйма. Арбитраж не разрешается между:

– повторным условием «СТАРТ» и битом данных;

– условием «СТОП» и битом данных;

– повторным условием «СТАРТ» и условием «СТОП».

**Режим ведомого**

В режиме ведомого операции передачи и приема управляются автоматически модулем I2C.

В режиме ведомого приемника биты данных принимаются на SDA и сдвигаются по тактовым импульсам, генерируемым ведущим устройством. Ведомое устройство не генерирует тактовый сигнал, но может удерживать линию SCL в состоянии низкого уровня, если после приема байта необходимо вмешательство ЦПУ.

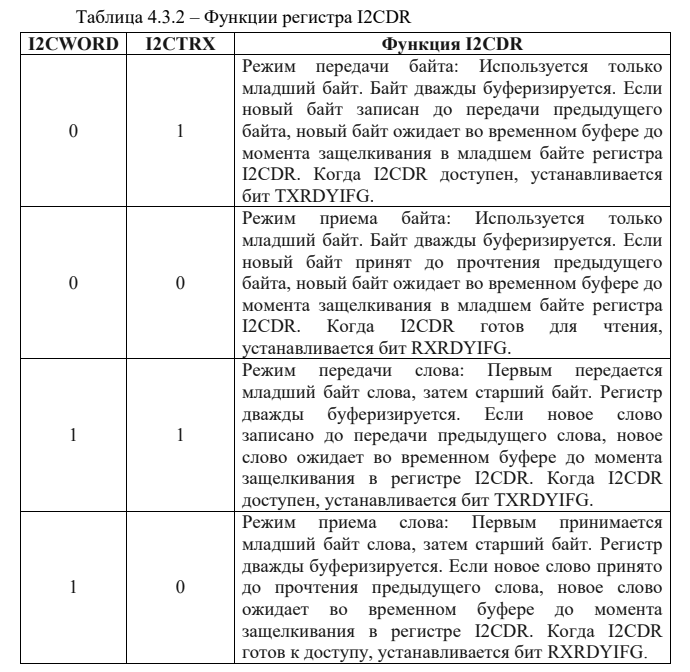
В режим ведомого передатчика можно войти только из режима ведомого приемника. Вход в режим ведомого передатчика происходит, если байт адреса ведомого, переданный ведущим, является таким же адресом, как и его собственный и был послан установленный бит C:\Users\500a5\Desktop\17-04-2021 16-21-55.png, указывая на запрос отправки данных ведущему. Ведомый передатчик сдвигает последовательные данные из устройства на SDA по импульсам тактирования, генерируемым ведущим устройством. Ведомое устройство не генерирует тактовых сигналов, но может удерживать линию SCL в состоянии низкого уровня, если после передачи байта необходимо вмешательство ЦПУ.

**Регистр данных I2CDR шины I2C Регистр I2CDR**

Регистр данных I2CDR шины I2C Регистр I2CDR может быть доступен как 8-разрядный или 16-разрядный регистр, что определяется битом I2CWORD. Функции регистра I2CDR описаны в таблице 4.3.2.

**Опустошение при передаче**

В режиме ведущего опустошение происходит, когда сдвиговый регистр передачи и буфер передачи пусты, а I2CNDAT > 0. В режиме ведомого опустошение происходит, когда сдвиговый регистр передачи и буфер передачи пусты, а внешний ведущий I2C все еще запрашивает данные. Когда происходит опустошение при передаче, устанавливается бит I2CTXUDF. Запись данных в регистр I2CDR или сброс бита I2CEN сбрасывает I2CTXUDF. I2CTXUDF используется только в режиме передачи.

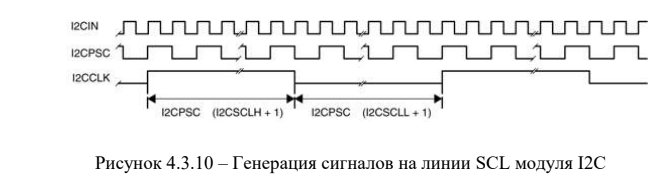


**Переполнение при приеме**

Переполнение при приеме происходит, когда сдвиговый регистр приема и буфер приема заполнены. Когда происходит переполнение при приеме, устанавливается бит I2CRXOVR. Потери данных не происходит, поскольку в этом случае линия SCL удерживается в состоянии низкого уровня, которое приостанавливает дальнейшую активность на шине. Чтение регистра I2CDR или сброс бита I2CEN сбрасывает бит I2CRXOVR. Бит I2CRXOVR используется только в режиме приема.

**Генерация тактовых сигналов I2C и синхронизация**

Модуль I2C работает с источником тактовой частоты, выбираемым битами I2CSSELx. Прескалер I2CPSC и регистры I2CSCLH и I2CSCLL определяют частоту и скважность тактового сигнала SCL для режима ведомого, как показано на рисунке 4.3.10. Источник тактовых импульсов для модуля I2C должен иметь частоту, по крайней мере в 10 раз больше частоты SCL в обоих режимах ведущего и ведомого.



**Конфигурирование USART для функционирования I2C**

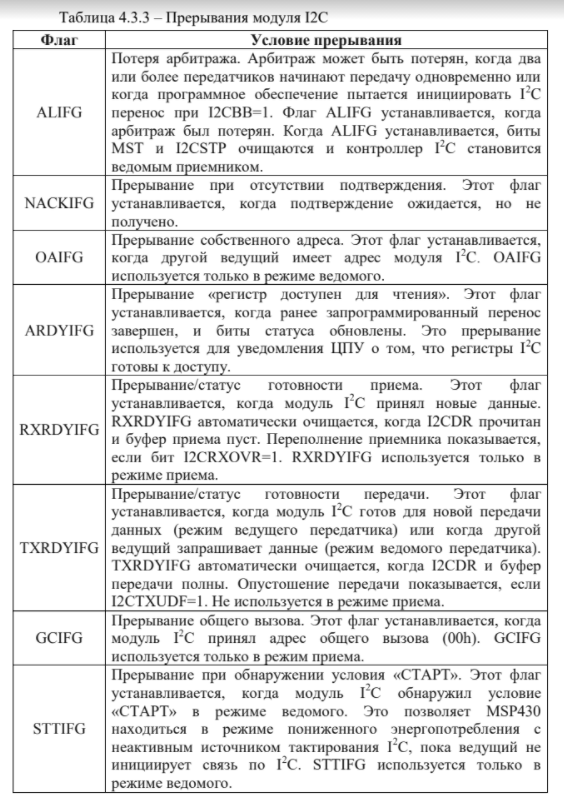
Контроллер I2C является частью периферии USART. Для работы в режимах SPI или I2C необходимо установить бит SYNC. Установка бита SYNC при SWRST=1 выбирает режим SPI. Установка бита I2C, когда SYNC=1 приводит к выбору режима I2C. Биты SYNC и I2C могут быть установлены вместе в одной команде для выбора режима I2C в модуле USART0. После инициализации модуль I2C готов для выполнения операций передачи и приема. Очистка I2CEN прекращает работу модуля.

**Прерывания I2C**

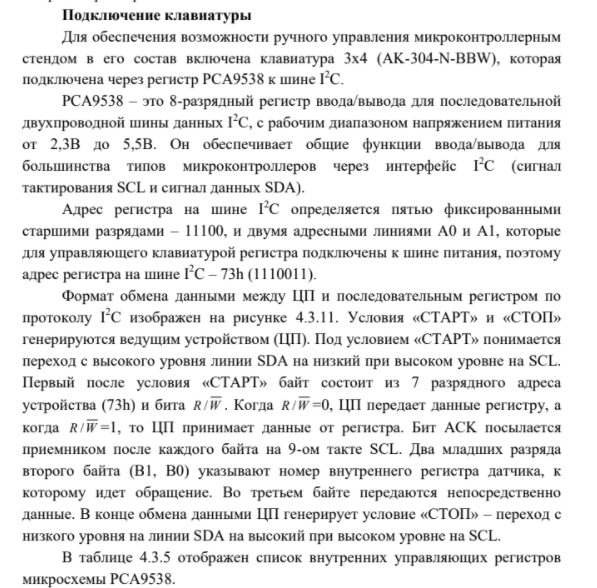
Модуль I2C имеет один вектор прерывания для восьми флагов прерывания. Каждый флаг прерывания имеет собственный бит разрешения прерывания. Когда прерывание разрешено и установлен бит GIE, флаг прерывания будет генерировать запрос прерывания. События, перечисленные в таблице 4.3.3, вызывают I2C прерывание.

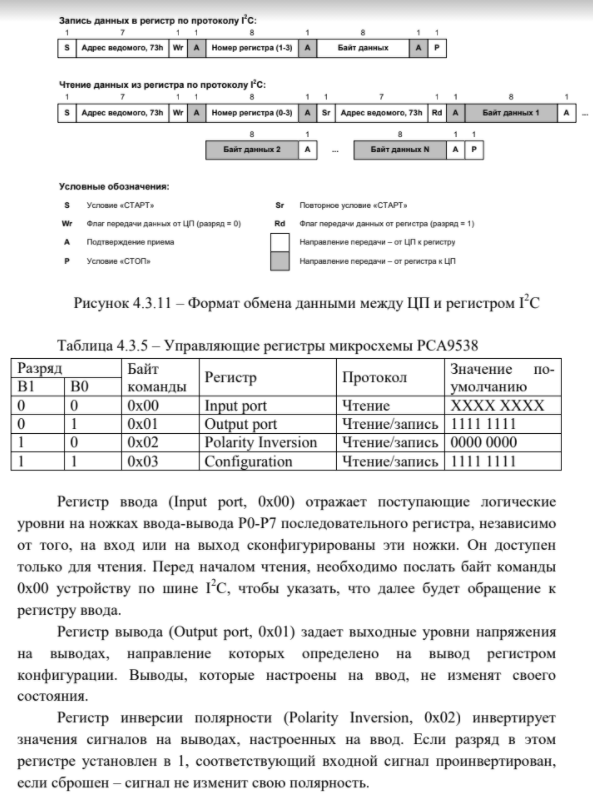
**Генератор вектора прерывания I2CIV**

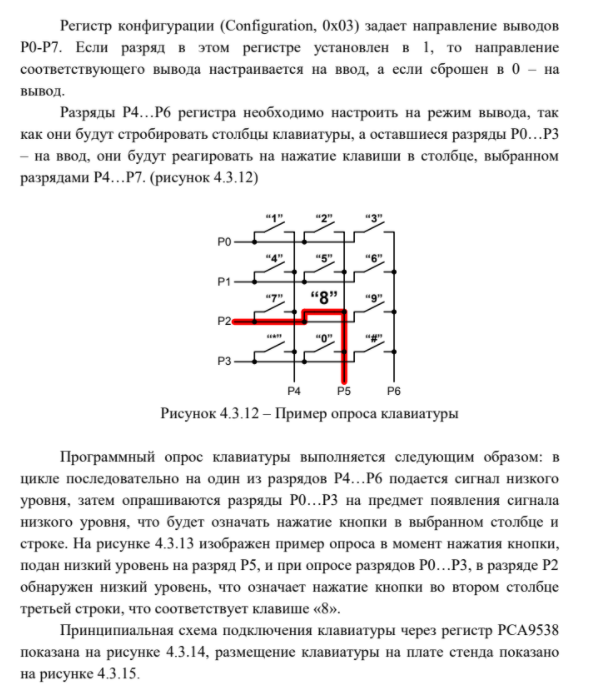
Флаги прерывания I2C разделены по приоритетам и объединены в источник одного вектора прерывания. Регистр вектора прерывания I2CIV используется для выяснения, какой флаг запросил прерывание. Разрешенное прерывание с наивысшим приоритетом генерирует число в регистре I2CIV. Это число может быть оценено или добавлено к программному счетчику для автоматического входа в соответствующую программную процедуру. Запрещенные I2C прерывания не воздействуют на содержимое I2CIV.



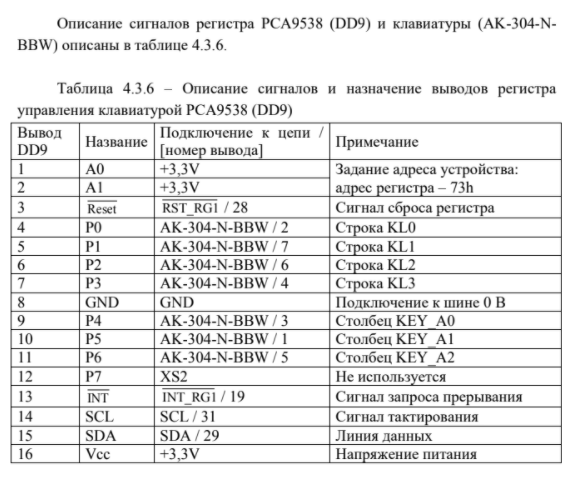
При любом доступе (чтение или запись) к регистру I2CIV автоматически сбрасывается флаг ожидающего прерывания с наивысшим приоритетом. Если устанавливается другой флаг прерывания, после обработки начального прерывания немедленно генерируется другое прерывание.

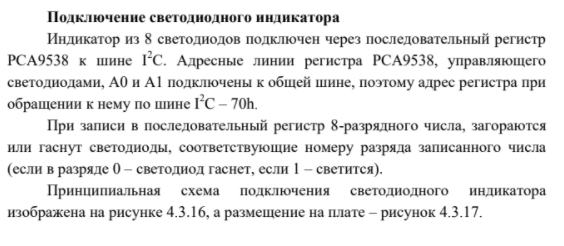




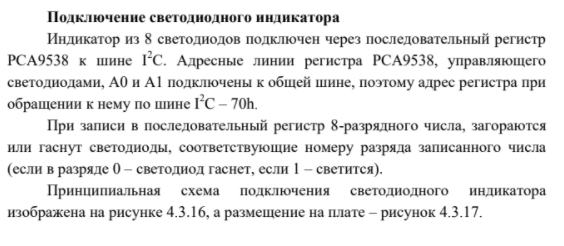




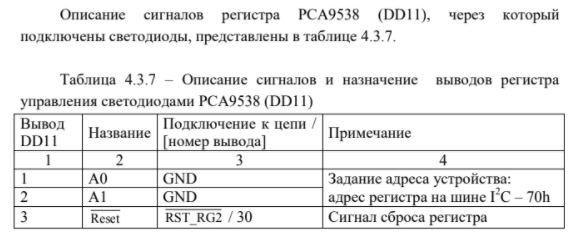


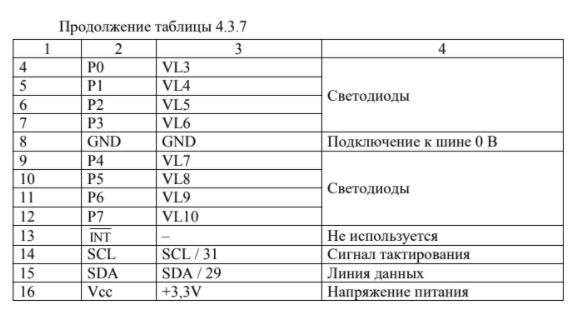












**Порядок выполнения задания:**

– включить лабораторный макет.

– запустить компилятор IAR Embedded Workbench.

– создать пустой проект.

– создать файл ресурса для кода программы и подключить его к проекту.

– ввести код исходного модуля программы обмена данными между микроконтроллером MSP430F1611 с регистрами PCA9538 по интерфейсу I2C соответствие с индивидуальным заданием, приведенным в таблице 4.3.8.

– выполнить компиляцию исходного модуля программы и устранить ошибки, полученные на данном этапе.

– настроить параметры программатора.

– создать загрузочный модуль программы и выполнить программирование микроконтроллера.

Проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.

В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы.

**Содержимое файла main.c**

#include <msp430.h>

#include "system\_define.h"

#include "system\_variable.h"

#include "function\_prototype.h"

#include "I2C.h"

#include "main.h"

/\*

\* main.c

\*/

void main(void) {

\_enable\_interrupt();

WDTCTL = WDTPW+WDTHOLD;

Init\_System\_Clock();

Init\_System();

Init\_I2C();

LED\_clear();

char key;

int f = 0;

Init\_I2C();

I2CSCLH = 0x26; //38 + 2 = 40

I2CSCLL = 0x26; //38 + 2 = 40

// 40\*2=80

// 8 000 000 / 80 = 30 000 Гц

while (f != 1) {

key = KEYS\_scannow();

switch (key) {

case '9':

LED\_clear();

LED\_set(1);

break;

case '0':

LED\_clear();

LED\_set(2);

break;

case '1':

LED\_clear();

LED\_set(3);

break;

case '\*':

f = 1;

}

}

while(1){

}

}

**Содержимое файла I2C.c**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// I2C function

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "function\_prototype.h"

#include "system\_define.h"

#include "I2C.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//============================================================================

// Инициализация модуля UART0 для работы в режиме I2C

void Init\_I2C()

{

P3SEL |= 0x0A; // Выбор альтернативной функции для линий порта P3

// в режиме I2C SDA->P3.1, SCL->P3.3

U0CTL |= I2C + SYNC; // Выбрать режим I2C для USART0

U0CTL &= ~I2CEN; // Выключить модуль I2C

// Конфигурация модуля I2C

I2CTCTL=I2CSSEL\_2; // SMCLK

I2CSCLH = 0x26; // High period of SCL

I2CSCLL = 0x26; // Low period of SCL

U0CTL |= I2CEN; // Включить модуль I2C

// формирование строба сброса I2C-регистров PCA9538 - RST\_RG1->P3.1 и RST\_RG2->P3.2

P3DIR |= 0x05; // переключаем эти ножки порта на вывод,

P3SEL &= ~0x05; // выбираем функцию ввода-вывода для них

P3OUT &= ~0x05; // и формируем строб сброса на 1 мс

wait\_1ms(1);

P3OUT |= 0x05;

}

// Инициализация модуля UART0 для работы в режиме I2C

void Init\_I2C\_custom\_speed()

{

P3SEL |= 0x0A; // Выбор альтернативной функции для линий порта P3

// в режиме I2C SDA->P3.1, SCL->P3.3

U0CTL |= I2C + SYNC; // Выбрать режим I2C для USART0

U0CTL &= ~I2CEN; // Выключить модуль I2C

// Конфигурация модуля I2C

I2CTCTL=I2CSSEL\_2; // Используем SMCLK = 8 МГц

// Необходимо получить частоту 20 кГц.

// Прескалер не стоит выставлять больше 4

I2CPSC = 0x03; // Прескалер

// I2CSCLH = I2CSCLL = 99 = 63h

I2CSCLH = 0x63; // Высокий уровень SCL: (I2CPSC+1)(I2CSCLH+1)

I2CSCLL = 0x63; // Низкий уровень SCL (I2CPSC+1)(I2CSCLH+1)

U0CTL |= I2CEN; // Включить модуль I2C

// формирование строба сброса I2C-регистров PCA9538 - RST\_RG1->P3.1 и RST\_RG2->P3.2

P3DIR |= 0x05; // переключаем эти ножки порта на вывод,

P3SEL &= ~0x05; // выбираем функцию ввода-вывода для них

P3OUT &= ~0x05; // и формируем строб сброса на 1 мс

wait\_1ms(1);

P3OUT |= 0x05;

}

//============================================================================

//============================================================================

// отправка данных по протоколу I2C

void Send\_I2C(unsigned char\* buffer,unsigned int num, unsigned char address)

{

while (I2CBUSY & I2CDCTL); // проверка готовности модуля I2C

BufTptr=buffer;

I2CSA = address; // установка адреса приемнмка

I2CNDAT =num; // количество передаваемых байт

I2CIE = TXRDYIE+ALIE; // разрешение прерываний по окончанию передачи байта и по потере арбитража

U0CTL |= MST; // режим Master

I2CTCTL |= I2CSTT + I2CSTP + I2CTRX; // инициализировать передачу

while ((I2CTCTL & I2CSTP) == 0x02); // ожидание условия СТОП

}

//============================================================================

//============================================================================

// прием данных по протоколу I2C

void Receive\_I2C(unsigned char\* buffer,unsigned int num, unsigned char address)

{

while (I2CBUSY & I2CDCTL); // проверка готовности модуля I2C

BufRptr=buffer;

I2CSA=address;

I2CTCTL&=~I2CTRX; // режим приема

I2CNDAT=num;

I2CIE=RXRDYIE; // резрешение прерывания по окончанию приема байта

U0CTL |= MST;

I2CTCTL |= I2CSTT + I2CSTP; // инициализировать прием

while ((I2CTCTL & I2CSTP) == 0x02); // ожидание условия СТОП

}

//============================================================================

//============================================================================

// отправка байта устройству на шине I2C

void I2C\_SendByte(char data, char i2c\_addr)

{

Tx\_Data[0] = data; // отправляемый байт

Send\_I2C(&Tx\_Data[0], 1, i2c\_addr); // вывод по I2C на устройство

}

//============================================================================

//============================================================================

// запись байта в регистр устройства на шине I2C

void I2C\_WriteByte(char reg, char data, char i2c\_addr)

{

Tx\_Data[0] = reg; // выбираем регистр

Tx\_Data[1] = data; // записываемые данные

Send\_I2C(&Tx\_Data[0], 2, i2c\_addr); // вывод по I2C на устройство

}

//============================================================================

//============================================================================

// чтение байта из регистра устройства на шине I2C

byte I2C\_ReadByte(char reg, char i2c\_addr)

{

Tx\_Data[0] = reg; // выбираем регистр

Send\_I2C(&Tx\_Data[0], 1, i2c\_addr);

Receive\_I2C(&Rx\_Data[0], 1, i2c\_addr); // получаем значение из регистра

return Rx\_Data[0];

}

//============================================================================

//============================================================================

// чтение слова (2 байта) из регистра устройства на шине I2C

int I2C\_ReadWord(char reg, char i2c\_addr)

{

Tx\_Data[0] = reg; // выбираем регистр

Send\_I2C(&Tx\_Data[0], 1, i2c\_addr);

Receive\_I2C(&Rx\_Data[0], 2, i2c\_addr); // получаем 2 байта значение из регистра

return Rx\_Data[0] + (Rx\_Data[1] \* 256);

}

//============================================================================

//============================================================================

//Обработка прерывания от модуля USART0, работающего в режиме I2C

// вектор прерываний для модуля I2C

#pragma vector=USART0TX\_VECTOR

\_\_interrupt void I2C\_ISR()

{

switch(I2CIV)

{

case 0: break; // нет прерывания

case 2: break; // потеря арбитража

case 4: break; // нет подтверждения

case 6: break; // прерывание собственного адреса

case 8: break; // регистр доступен для чтения

case 10: // окончание приема байта

\*BufRptr++=I2CDRB;

break;

case 12: // окончание передачи байта

I2CDRB=\*BufTptr++;

break;

case 14: break; // общий вызов

case 16: break; // обнаружено условие СТАРТ

default : break;

}

}//============================================================================

//-------------------------------------------------

**Содержимое файла keys.c**

// Keyboard functions

#include "function\_prototype.h"

#include "sysfunc.h"

#include "keys.h"

byte keycol, keyline, KEYS\_last=0;

char table\_keys[12] = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '\*', '0', '#'};

// Проверка нажатия клавиши в текущий момент, результат:

// 0 - клавиша не нажата

// ASCII-код клавиши

char KEYS\_scannow()

{

keyline=0;

// выбираем регистр конфигурации направления (0x03)

// и конфигурируем P4-P7 на вывод - для строба столбцов,

// а P0-P3 на ввод - для опроса строк (1-ввод, 0-вывод)

I2C\_WriteByte(0x03, 0x0F, KEYS\_i2c\_addr);

for (keycol=0; keycol<3; keycol++) {

// последовательно подаем сигнал низкого уровня на столбцы (P4-P7)

I2C\_WriteByte(0x01, ~(1<<keycol<<4) & 0xf0, KEYS\_i2c\_addr);

wait\_1ms(1);

// и опрашиваем строки (P0-P3) на наличие нуля

keyline = ~(I2C\_ReadByte(0x00, KEYS\_i2c\_addr)) & 0x0f;

if (keyline) break;

}

if (!keyline) return 0; // если не была нажата никакая клавиша - возвращаем 0

if (keyline == 4) keyline = 3; // переводим номера разрядов в номер строки

if (keyline == 8) keyline = 4;

KEYS\_last = table\_keys[--keyline\*3+keycol]; // получаем код нажатой клавиши из таблицы

return KEYS\_last;

}

// Возвращает код последней нажатой клавиши, результат:

// 0 - не нажималась никакая клавиша

// ASCII-код клавиши

char KEYS\_lastkey()

{

KEYS\_scannow();

return KEYS\_last;

}

// Очистка последней нажатой клавиши

void KEYS\_clear()

{

KEYS\_last = 0;

wait\_1ms(200);

}

// Ожидание нажатия клавиши, результат - ASCII-код нажатой клавиши

char KEYS\_waitkey()

{

KEYS\_clear(); // очистка последней нажатой клавиши

while (!KEYS\_scannow()) // пока не нажата никакая клавиша,

wait\_1ms(1); // сделать паузу

return KEYS\_last; // вернуть код нажатой клавиши

}

// пауза с циклическим опросом клавиатуры, прерывается если нажата клавиша

void KEYS\_pause(byte cnt)

{

byte i;

KEYS\_clear(); // очистка последней нажатой клавиши

for (i=0; i<cnt; i++)

if (KEYS\_scannow())

break;

}

**Содержимое файла leds.c**

// LED-indicator functions

#include "function\_prototype.h"

#include "sysfunc.h"

#include "leds.h"

char LED\_config=0; // хранится конфигурация светодиодов (вкл/выкл)

void LED\_out(char leds)

{

// регистр конфигурации направления 0x03 конфигурируем на вывод информации (1-ввод, 0-вывод)

I2C\_WriteByte(0x03, 0x00, LED\_i2c\_addr);

I2C\_WriteByte(0x01, leds, LED\_i2c\_addr); // выводим данные в регистр OUTPUT (0x01)

LED\_config=leds; // сохраняем новую конфигурацию

}

// Преобразование номера светодиода в бит, с которым нужно проводить операцию

// 1 = 10000000, 2 = 01000000 ... 8 = 00000001

char LED\_convert(char led)

{

if(led<1)

led=1;

if(led>8)

led=8;

led=9-led;

return (1<<(led-1));

}

// Выключить все светодиоды

void LED\_clear()

{

LED\_out(0x00);

}

// Инвертировать все светодиоды

void LED\_invert()

{

LED\_out(LED\_config ^ 0xff);

}

// Включить светодиод с номером от 1 до 8 (слева направо)

void LED\_set(char led)

{

led=LED\_convert(led);

LED\_config |= led; // устанавливаем соответствующий разряд

LED\_out(LED\_config); // выводим в регистр

}

// Выключить светодиод с номером от 1 до 8 (слева направо)

void LED\_reset(char led)

{

led=LED\_convert(led);

LED\_config &= ~(led); // сбрасываем соответствующий разряд

LED\_out(LED\_config); // выводим в регистр

}

// Сменить состояние светодиода с номером от 1 до 8 (слева направо)

void LED\_change(byte led)

{

led=LED\_convert(led);

LED\_config ^= led; // меняем состояние соответствующего разряда (XOR)

LED\_out(LED\_config); // выводим в регистр

}

void LED\_fx1(int n)

{

LED\_clear();

LED\_set(1);

wait\_1ms(n);

LED\_set(3);

wait\_1ms(n);

LED\_set(5);

wait\_1ms(n);

LED\_set(7);

wait\_1ms(n);

LED\_reset(1);

wait\_1ms(n);

LED\_reset(3);

wait\_1ms(n);

LED\_reset(5);

wait\_1ms(n);

LED\_reset(7);

wait\_1ms(n);

LED\_clear();

}

void LED\_fx2(int n)

{

LED\_clear();

LED\_out(0x81);

wait\_1ms(n);

LED\_out(0x42);

wait\_1ms(n\*8);

LED\_out(0x24);

wait\_1ms(n\*5);

LED\_out(0x18);

wait\_1ms(n\*3);

LED\_out(0x24);

wait\_1ms(n\*5);

LED\_out(0x42);

wait\_1ms(n\*8);

LED\_clear();

}

void LED\_fx3(int n)

{

LED\_clear();

LED\_out(0x01);

wait\_1ms(n\*6);

LED\_out(0x02);

wait\_1ms(n\*4);

LED\_out(0x04);

wait\_1ms(n\*2);

LED\_out(0x08);

wait\_1ms(n\*1);

LED\_out(0x10);

wait\_1ms(n\*1);

LED\_out(0x20);

wait\_1ms(n\*2);

LED\_out(0x40);

wait\_1ms(n\*4);

LED\_out(0x80);

wait\_1ms(n\*6);

LED\_out(0x40);

wait\_1ms(n\*4);

LED\_out(0x20);

wait\_1ms(n\*2);

LED\_out(0x10);

wait\_1ms(n\*1);

LED\_out(0x08);

wait\_1ms(n\*1);

LED\_out(0x04);

wait\_1ms(n\*2);

LED\_out(0x02);

wait\_1ms(n\*4);

LED\_clear();

}

**Содержимое файла sysfunc.c**

// System functions

#include <msp430.h>

#include "sysfunc.h"

// инициализация портов системы

void Init\_System()

{

P1DIR |= (nSS + nWR\_nRST + MCU\_SEL\_0 + MCU\_SEL\_1); // установка направления портов на вывод

DB\_DIR = 0x00; // шина данных настроена на ввод

}

// инициализация системы тактирования

void Init\_System\_Clock()

{

volatile byte i;

BCSCTL1 &= ~XT2OFF; // включение осцилятора XT2

// MCLK = XT2, SMCLK = XT2

do // ожидание запуска кварца

{

IFG1 &= ~OFIFG; // Clear OSCFault flag

for (i = 0xFF; i > 0; i--); // Time for flag to set

}

while ((IFG1 & OFIFG)); // OSCFault flag still set?

BCSCTL2 |= SELM\_2 | SELS; // установка внешнего модуля тактирования

}

// 2do: сделать точную задержку

void wait\_1ms(word cnt)

{

for (wait\_i = 0; wait\_i < cnt; wait\_i++)

for (wait\_j = 0; wait\_j < 1000; wait\_j++);

}

void wait\_1mks(word cnt)

{

for (wait\_i = 0; wait\_i < cnt; wait\_i++);

}

**Содержимое файла msp430\_initialize\_unused\_interrrupt\_vectors.c**

#include <msp430.h>

/\* Initialize non-used ISR vectors with a trap function \*/

/\*

DACDMA\_VECTOR (0 \* 1u) 0xFFE0 DAC/DMA

PORT2\_VECTOR (1 \* 1u) 0xFFE2 Port 2

USART1TX\_VECTOR (2 \* 1u) 0xFFE4 USART 1 Transmit

USART1RX\_VECTOR (3 \* 1u) 0xFFE6 USART 1 Receive

PORT1\_VECTOR (4 \* 1u) 0xFFE8 Port 1

TIMERA1\_VECTOR (5 \* 1u) 0xFFEA Timer A CC1-2, TA

TIMERA0\_VECTOR (6 \* 1u) 0xFFEC Timer A CC0

ADC12\_VECTOR (7 \* 1u) 0xFFEE ADC

USART0TX\_VECTOR (8 \* 1u) 0xFFF0 USART 0 Transmit // Прерывание используется

USART0RX\_VECTOR (9 \* 1u) 0xFFF2 USART 0 Receive // Прерывание используется

WDT\_VECTOR (10 \* 1u) 0xFFF4 Watchdog Timer

COMPARATORA\_VECTOR (11 \* 1u) 0xFFF6 Comparator A

TIMERB1\_VECTOR (12 \* 1u) 0xFFF8 Timer B CC1-6, TB

TIMERB0\_VECTOR (13 \* 1u) 0xFFFA Timer B CC0

NMI\_VECTOR (14 \* 1u) 0xFFFC Non-maskable

RESET\_VECTOR (15 \* 1u) 0xFFFE Reset [Highest Priority]

\*/

#pragma vector=DACDMA\_VECTOR,PORT2\_VECTOR,USART1TX\_VECTOR,USART1RX\_VECTOR,TIMERA1\_VECTOR, \

TIMERA0\_VECTOR,ADC12\_VECTOR,WDT\_VECTOR,COMPARATORA\_VECTOR,TIMERB1\_VECTOR,TIMERB0\_VECTOR, \

NMI\_VECTOR,RESET\_VECTOR

\_\_interrupt void ISR\_trap(void)

{

// Вызов неиспользуемого прерывания приведет к сбросу процессора (PUC reset)

WDTCTL = 0;

}

**Выводы:** изучил принципы программного управления двунаправленным обменом данных по последовательному интерфейсу I2C.